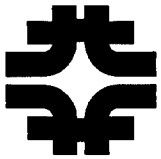

Fermi National Accelerator Laboratory

FERMILAB-Conf-94/092

DBS—An rlogin Multiplexor and Output Logger for DA Systems

G. Oleynik, L. Appleton, L. Udumula, M. Votava

Fermi National Accelerator Laboratory
P.O. Box 500, Batavia, Illinois 60510

April 1994

Presented at the *Conference on Computing in High Energy Physics 94*,
San Francisco, California, April 21-27, 1994

Operated by Universities Research Association Inc. under Contract No. DE-AC02-76CHO3000 with the United States Department of Energy

Disclaimer

DBS - An rlogin Multiplexor and Output Logger for DA Systems*

Gene Oleynik, Laura Appleton, Lourdu Udumula, Margaret Votava
Online Systems Department
Fermi National Accelerator Laboratory
P.O. Box 500
Batavia, IL 60510

Abstract

DART Bootstrap Services (**dbs**) is the first component of run-control for the DART Data Acquisition system - the DA for the 96' round of experiments at Fermilab - though it has potential usefulness as a powerful tool in other distributed applications.

dbs is an rlogin session multiplexer. It allows a user, running a single program, to start up any number of remote login sessions, feed shell commands to them, and collect their output into a single (or multiple) log files (a server keeps the sessions open and collects their output). From this program, any session can be attached to interactively so it appears just like an rlogin session - **dbs** becomes transparent. When finished with this interactive mode, the user can escape back to **dbs** and attach to a different session if so desired. Among many other useful features, **dbs** supplies a mechanism for cleanup (deletion) of all processes created under a session, allowing a fresh start.

1 Introduction

dbs is a Unix based tool designed to meet certain needs for working with a fully distributed system. Though it was developed for DART data acquisition systems, its domain of applicability goes beyond this specific application. We have taken care that the software product is generic enough to leverage its use across a variety of potential applications.

2 Motivation and Requirements

DART data acquisition systems consist of a number of intelligent computers, running a number of operating systems and a variety of applications, distributed across a LAN [1,2]. Many of these applications are replicated across nodes. Taken together, these cooperating applications make up a DA system. The specific applications that make up the DA and the architecture of the system, vary from experiment to experiment. **dbs** addresses the problem of how to start-up applications for a given experiment in a uniform manner such that the system comes up correctly as a whole, or is "bootstrapped".

There are several pertinent factors for the solution of this problem that we folded into **dbs** requirements [3]:

- It must be able to start-up all applications for the system from a source file on a single node with minimal user interaction.
- It must be easily tailorable with different sets of applications for different systems.
- It must be able to start-up any application. There should not have to be anything special added to an application.
- It must handle replication of applications efficiently.
- It must provide some mechanism to synchronize the start of client-server applications.
- It must be able to start-up applications on multiple operating systems, including Unix, VMS, and VxWorks (the real-time operating system we use on embedded processors), and be easily portable to oth-

* This work is sponsored by DOE contract No. DE-AC02-76CH03000

ers.

- It must be able to start applications in parallel across nodes so that the system can be bootstrapped quickly.
- It must provide a mechanism to remove all applications so that a clean start can be made.
- It must record information so that start-up failures are diagnosable, and collect the standard output and standard error from applications.
- It must use as little system resources as possible, and where possible, make use of existing system based software

3 dbs

We found the most effective solution to the bootstrap problem that meets the above requirements to be an application built on top of the ubiquitous, commercial network package, rlogin, which allows authorized users to login to a system without a password. rlogin software is bundled with the system on all Unix platforms and is available under VMS (e.g. via MULTINET) and VxWorks, as well as many other systems. In essence, no remote software needed to be written for **dbs**. **dbs** consists mainly of "host" software that runs on a single node to "launch" applications through rlogin connections to other nodes From here on, "host" will refer to the computer on which **dbs** runs to launch applications, and "remote" will refer to nodes on which the applications are started, which can include the host node.

The host **dbs** software consists of an rlogin multiplexing server and a user interface application, referred to as the "client", which drives this server [4]. Through the client, the user can create named rlogin "sessions" to remote nodes, non-interactively send commands to them to be executed, or connect "transparently" to any one of the sessions. In the latter case, the **dbs** client is transparent to the user, that is, it appears as a normal rlogin to the remote host, except that an "escape" breaks the user back to the **dbs** client. The client uses the TCL command line interpreter (free software) [5] and can be driven from a command file or interactively. The **dbs** client program can be exited at any time, and restarted later - the server keeps the sessions that are open and available.

The rlogin multiplexor server manages the bookkeeping of the sessions and routes session input and output between the remote sessions and the client application, or, as will be seen later, it can route session output to files on the host.

4 Session Management and Commands

A **dbs** user creates an rlogin session to a remote node with a command that associates a "session-name" with the session. The session-name is used to identify the session in all further transactions. As will be seen, this simple session-name abstraction has powerful consequences.

Commands are sent to a session with the "s_command" client command. This command is asynchronous in the sense that it is sent to the sessions standard input without waiting for an acknowledgment, permitting commands to be executed on different sessions in parallel. A command can be broadcast to all sessions by leaving the session-name field blank; we have plans for more powerful wildcarding.

5 Session Output Management & Logging

The only feedback from remote sessions is their output through standard out and standard error. One implication of the requirement that **dbs** be able to start "any old application" is that **dbs** has no control over its output, which may contain important information. Therefore, **dbs** was designed to optionally capture session output and write it to a file.

dbs buffers the output from its sessions in the server. The buffer size can be specified per session when the session is created (a default is provided).

If no output file is specified for the session, with the s_fileout command, then if the ses-

sion's buffer fills, the output will back up to the remote session and eventually block the application producing it, until the session is attached to interactively through **dbs**, or an output file is specified for the session.

If an output file is specified for a session, then the sessions output buffer is flushed to the file periodically at a user specified interval. The output is also automatically flushed to the output file if the session's buffer fills.

Any number of sessions' output can be directed to a single file; the s_fileout command takes the "all sessions" wildcarding. A time and session-name stamp precedes each flush so one can make sense of the mixed session output.

6 Remote Session Software Library - Application Cleanup and Synchronization

We have strived to keep the remote software to a minimum. **dbs** can be run with no special software on the remote side. However, in order to be able to use the functionality of cleaning up (i.e. killing or removing) applications for a clean start, or synchronizing the start-up of a server and its clients, some remote software is required.

We provide a process "cleanup" registration routine that applications call if they need to be killed via the "s_cleanup" command. Registration and cleanup is done on a per session basis, but like other **dbs** commands, s_cleanup accepts wildcarding. Registration is done through foolproof mechanisms, such as "fuser" on Unix and a task variable mechanism developed for VxWorks.

Some applications need to reach a certain point before others can be started. An example is a network client-server application, for which the server needs to begin listening for client connections before the clients can be started. **dbs** provides the "s_waitcommand" command for this purpose. s_waitcommand sends a command to the remote session, and waits with a time-out for a special character sequence to be sent back over the standard out of the remote session. A routine is provided for remote applications to write this sequence to their standard output. In the client-server example, the server would call this routine immediately before listening for client connections, and the clients would be started immediately after the server.

7 Security

Since **dbs** is rlogin based, it inherits rlogin security of account based .rhosts, in which trusted machine/account pairs are specified in this file. Account level security is integrated into **dbs**.

8 Future Plans

We currently plan to add two new features: routing session output to slave Xterm windows in addition to files and supporting full regular expression wildcarding for sessions. The former will add more visual monitoring of session output, and the latter the command multicasting capability, e.g. sending a command to all nodes of a given operating system type (by embedding the OS type in the session-name), as well as directing output from sessions matching the wildcarding to a single file.

We wish to integrate **dbs** with our DA configuration management "database", so that the systems application start-up is specified along with other DA parameters through the database. Since **dbs** can take command input through a pipe, we imagine this will be accomplished by piping into **dbs** the output of an application which extracts the start-up information from the database.

We also plan to upgrade the command parsing of the client TCL commands to an improved parser we have developed.

9 Conclusions and Observations

One experience we would like to pass on is the fact that we believe we gained in productivity and maintainability by writing the multiplexing server in C++, and using a commercial class library, tools.h++, from Rogue Wave Inc. We were able to use their hash dictionary class for session-name mapping, and its regular expression capabilities will make it easy to extend the session-name wildcarding capabilities.

We have built a networking tool, **dbs**, to aid in the bootstrapping of systems built from distributed, cooperating applications. While we built this tool specifically for the needs of HEP data acquisition, we feel that we were careful enough to make the tool general enough for use in other distributed applications.

References

[1] R. Pordes et al., "Fermilab's DART DA System," these proceedings.

[2] G. Oleynik et al., "DART - Data Acquisition for the Next Generation of Fermilab Fixed Target Experiments," IEEE Transactions on Nuclear Science, Vol 41, No 1.

[3] L. Appleton et al., "DART Run Control Requirements," Fermilab Computing Division document PN-471.

[4] L. Appleton et al., "DART Bootstrap Services (**dbs**)," Fermilab Computing Division document PN-483.

[5] Ousterhout, "Tcl: An Embeddable Command Language," 1990 Winter USENIX Conference Proceedings.